Lab Manual

for

Linear Algebra

by

Jim Hefferon

Cover: my Chocolate Lab, Suzy.

Preface

WARNING! This is an incomplete draft, no doubt riddled with errors.

This collection supplements the text *Linear Algebra*[1] with explorations to help students solidify and extend their understanding of the subject, using the mathematical software Sage.[2]

A major goal of any undergraduate Mathematics program is to move students toward a higher-level, more abstract, grasp of the subject. For instance, Calculus classes work on elaborate computations while later courses spend more effort on concepts and proofs, working less on the details of calculations.

The text *Linear Algebra* fits into this development process. Naturally it presents the material using examples and practice problems that are small-sized and have manageable numbers: an assignment to by-hand multiply a pair of three by three matrices of small integers will build intuition, whereas asking students to do that same question with twenty by twenty matrices of ten decimal place numbers would be badgering.

However, an instructor can be concerned that this misses a chance to make the point that Linear Algebra is widely applied, or to develop students's understanding through explorations that are not hindered by the mechanics of paper and pencil. Mathematical software can mitigate these concerns by extending the reach of what is reasonable to bigger systems, harder numbers, and longer computations. For instance, an advantage of learning how to handle larger jobs is that they are more like the ones that appear when students apply Linear Algebra to other areas. Another advantage is that students see new ideas such as runtime growth measures.

Well then, why not teach straight from the computer?

Our goal is to develop a higher-level understanding of the material so we want to keep the focus on vector spaces and linear maps. This exposition has the computations as a way to develop that understanding, not as the main point.

Some instructors may find that their students are best served by keeping a tight focus on the core material, and leaving aside altogether the work in this manual. Other instructors have students who will benefit from the increased reach that the software provides. This manual's existence, and status as a separate book, gives teachers the freedom to make the choice that suits their class.

[1] The text's home page http://joshua.smcvt.edu/linearalgebra has the PDF, the ancillary materials, and the LaTeX source.
[2] See http://www.sagemath.org for the software and documentation.

Why Sage?

In *Open Source Mathematical Software* [Joyner and Stein, 2007][1] the authors argue that for Mathematics the best way forward is to use software that is Open Source.

> Suppose Jane is a well-known mathematician who announces she has proved a theorem. We probably will believe her, but she knows that she will be required to produce a proof if requested. However, suppose now Jane says a theorem is true based partly on the results of software. The closest we can reasonably hope to get to a rigorous proof (without new ideas) is the open inspection and ability to use all the computer code on which the result depends. If the program is proprietary, this is not possible. We have every right to be distrustful, not only due to a vague distrust of computers but because even the best programmers regularly make mistakes.
>
> If one reads the proof of Jane's theorem in hopes of extending her ideas or applying them in a new context, it is limiting to not have access to the inner workings of the software on which Jane's result builds.

Professionals choose their tools by balancing many factors but this argument is persuasive. We use Sage because it is very capable so students can learn a great deal from it, and because it is Free[2] and Open Source.[3]

This manual

This is Free. Get the latest version from http://joshua.smcvt.edu/linearalgebra. Also see that page for the license details and for the LaTeX source. I am glad to get feedback, especially from instructors who have class-tested the material. My contact information is on the same page.

The computer output included here was generated automatically (I have automatically edited some lines). This is my Sage.

```
'Sage Version 5.2, Release Date: 2012-07-25'
```

Acknowledgements

I am glad for this chance to thank the Sage Development Team. In particular, without [Sage Development Team, 2012b] this work would not have happened. I am glad also for the chance to mention [Beezer, 2011] as an inspiration.

We emphasize practice.
–Shunryu Suzuki [2006]

[A]n orderly presentation is not necessarily bad but by itself may be insufficient.
–Ron Brandt [1998]

Jim Hefferon
Mathematics, Saint Michael's College
Colchester, Vermont USA
2012-Sep-10

[1]See http://www.ams.org/notices/200710/tx071001279p.pdf for the full text. [2]The Free Software Foundation page http://www.gnu.org/philosophy/free-sw.html gives background and a definition. [3]See http://opensource.org/osd.html for a definition.

Contents

Singular Value Decomposition

Recall that a line through the origin in $\mathbb{R}^n$ is $\{r\cdot\vec{v} \mid r\in\mathbb{R}\}$. One of the defining properties of a linear map is that $h(r\cdot\vec{v}) = r\cdot h(\vec{v})$. So the action of h on any line through the origin is determined by the action of h on any nonzero vector in that line.

For instance consider the line $y=2x$ in the plane.

$$\{r\cdot\begin{pmatrix}1\\2\end{pmatrix} \mid r\in\mathbb{R}\}$$

If $t\colon\mathbb{R}^2\to\mathbb{R}^2$ is represented by the matrix

$$\mathrm{Rep}_{E_2,E_2}(t) = \begin{pmatrix}1&2\\3&4\end{pmatrix}$$

then here is the effect of t on one vector in the line.

$$\begin{pmatrix}1\\2\end{pmatrix} \overset{t}{\longmapsto} \begin{pmatrix}7\\10\end{pmatrix}$$

And here is the effect of t on other members of the line.

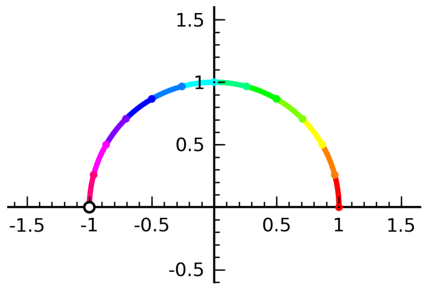
$$\begin{pmatrix}2\\4\end{pmatrix} \overset{t}{\longmapsto} \begin{pmatrix}14\\20\end{pmatrix} \qquad \begin{pmatrix}-3\\-6\end{pmatrix} \overset{t}{\longmapsto} \begin{pmatrix}-21\\-30\end{pmatrix} \qquad \begin{pmatrix}r\\2r\end{pmatrix} \overset{t}{\longmapsto} \begin{pmatrix}7r\\10r\end{pmatrix}$$

The map t's action on the line is uniform.

Unit circle

The above means that one way to describe a transformation's action is to pick one nonzero element from each line through the origin and describe where the transformation maps those elements. An easy way to select one nonzero element from each line through the origin is to take the upper half unit circle.

$$U = \{\begin{pmatrix}x\\y\end{pmatrix} \mid x=\cos(t),\ y=\sin(t),\ 0\leqslant t<\pi\}$$

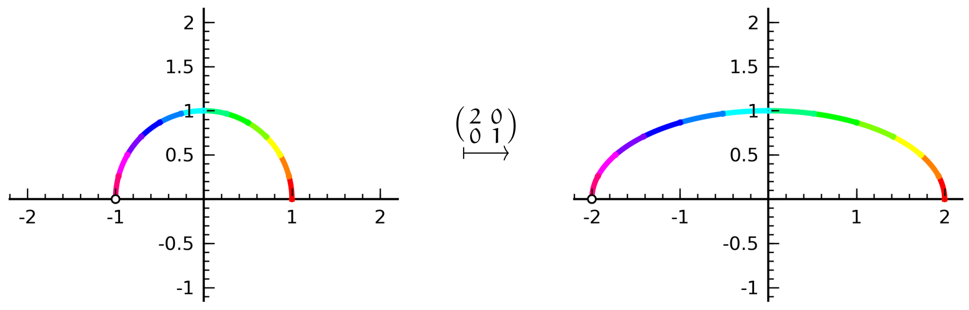


We produce the above graph by using a routine that draws the effect of an arbitrary matrix on the unit circle, and giving that routine the identity matrix. The colors will keep track the directionality in the set of before and after pictures below.

```
sage: load "plot_action.sage"
sage: p = plot_circle_action(1,0,0,1)  # identity matrix
sage: p.set_axes_range(-1.5, 1.5, -0.5, 1.5)
sage: p.save("sageoutput/plot_action10.png")
```

Here is the transformation doubles the x components of all vectors.

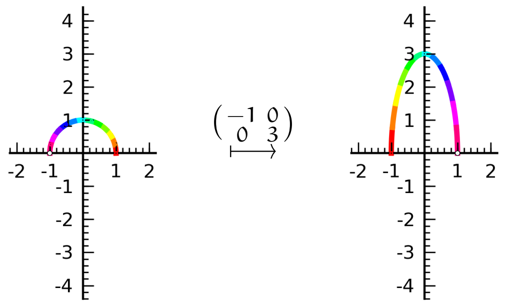
```
sage: p = plot_circle_action(2,0,0,1)
```



And here is the transformation that triples the y components and multiplies x components by -1.

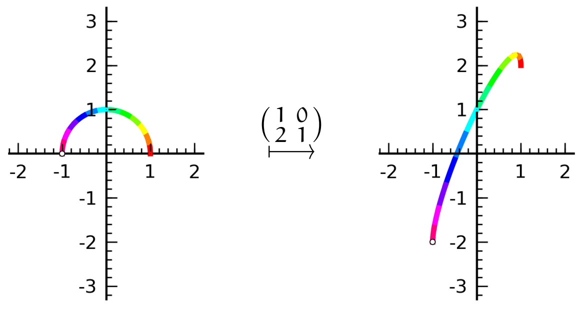
Note that it changes the orientation: the input circle moves counterclockwise from red to orange, then green, blue, indigo, and violet but the output does the opposite.

```
sage: p = plot_circle_action(-1,0,0,3)
```



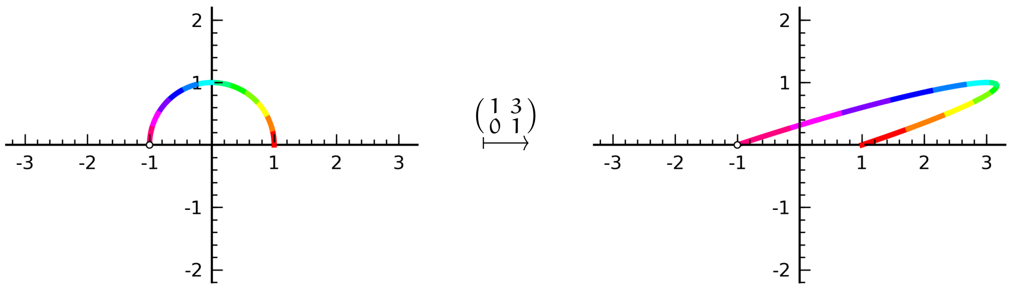
Here is the first skew transformation.

```
sage: p = plot_circle_action(1,0,2,1)
```



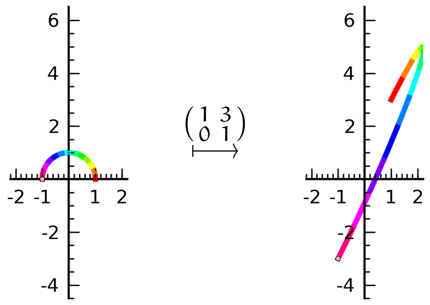
Here is the second skew transformation.

```
sage: p = plot_circle_action(1,3,0,1)
```



And here is the generic map.

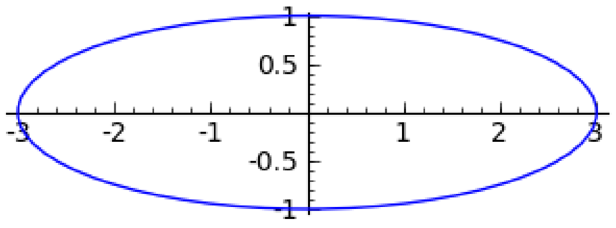
```
sage: p = plot_circle_action(1,2,3,4)
```



SVD

The above pictures show the unit circle mapping to ellipses. Recall that in $\mathbb{R}^2$ an ellipse has a *major axis*, the longer one, and a *minor axis*.[1] Write σ_1 for the length of the semi-major axis, the distance from the center to the furthest-away point on the ellipse, and write σ_2 for the length of the semi-minor axis.

```
sage: sigma_1=3
sage: sigma_2=1
sage: E = ellipse((0,0), sigma_1, sigma_2)
sage: E.save("sageoutput/svd01.png", figsize=3.5)
```

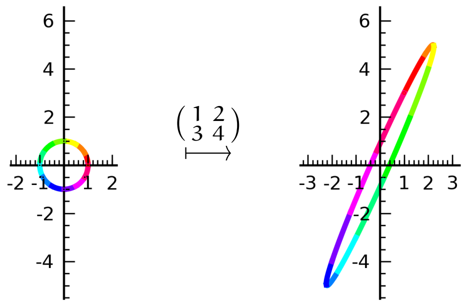


[1] If the two axes have the same length then the ellipse is a circle. If one axis has length zero then the ellipse is a line segment and if both have length zero then it is a point.

The two axes are orthogonal. In the above graph the major axis lies along the x-axis while the minor axis lies along the y-axis.

Under any linear map $t\colon \mathbb{R}^n \to \mathbb{R}^m$ the unit sphere maps to a hyperellipse. This is a version of the *Singular Value Decomposition* of matrices: for any linear map $t\colon \mathbb{R}^m \to \mathbb{R}^n$ there are bases $B = \langle \vec{\beta}_1, \dots, \vec{\beta}_m \rangle$ for the domain and $D = \langle \vec{\delta}_1, \dots, \vec{\delta}_n \rangle$ for the codomain such that $t(\vec{\beta}_i) = \sigma_i \vec{\delta}_i$, where the *singular values* σ_i are scalars. The next section states that result more fully and sketches the proof but first we illustrate the central idea by using the generic matrix. Here is its action again, this time on a full circle.

```
sage: p = plot_circle_action(1,2,3,4,full_circle=True)
```



(∗)

Sage will find the SVD of this example matrix.

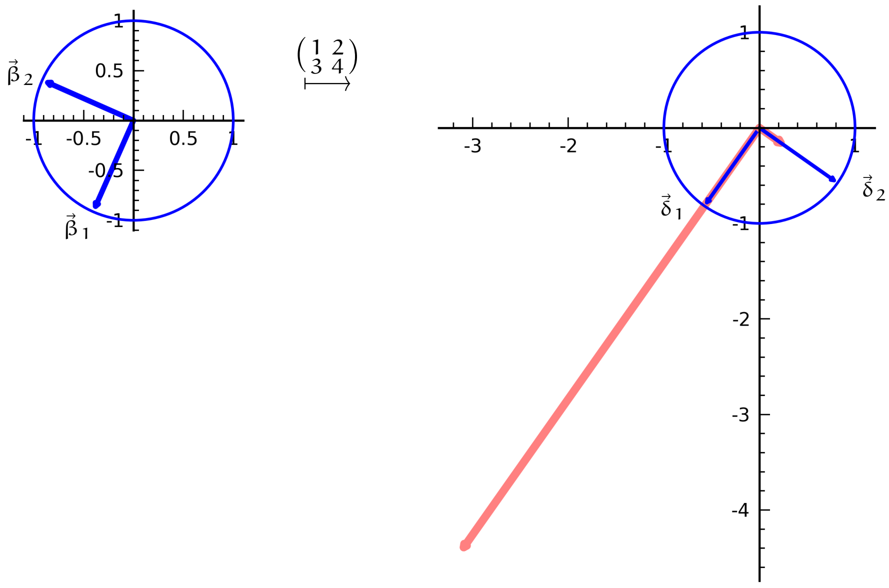
```
sage: M = matrix(RDF, [[1, 2], [3, 4]])
sage: U,Sigma,V = M.SVD()
sage: U
[-0.404553584834 -0.914514295677]
[-0.914514295677  0.404553584834]
sage: Sigma
[ 5.46498570422             0.0]
[           0.0  0.365966190626]
sage: V
[-0.576048436766   0.81741556047]
[  -0.81741556047 -0.576048436766]
sage: U*Sigma*(V.transpose())
[1.0 2.0]
[3.0 4.0]
```

The Singular Value Decomposition describes M as the product of three matrices, $U\Sigma V^T$. The basis vectors $\vec{\beta}_1$, $\vec{\beta}_2$, $\vec{\delta}_1$, and $\vec{\delta}_2$ are the columns of U and V. The singular values are the diagonal entries of Σ. Sage will plots the effect of the transformation on the basis vectors for the domain so we can compare those with the basis vectors for the codomain.

```
sage: beta_1 = vector(RR, [U[0][0], U[1][0]])
sage: beta_2 = vector(RR, [U[0][1], U[1][1]])
sage: delta_1 = vector(RR, [V[0][0], V[1][0]])
sage: delta_2 = vector(RR, [V[0][1], V[1][1]])
sage: plot.options['axes_pad'] = 0.05
sage: plot.options['fontsize'] = 7
sage: plot.options['dpi'] = 600
sage: plot.options['aspect_ratio'] = 1
```

```
sage: C = circle((0,0), 1)
sage: P = C + plot(beta_1) + plot(beta_2)
sage: P.save("sageoutput/svd02a.png", figsize=2, axes_pad=0.05)
sage: image_color=Color(1,0.5,0.5)   # color for t(beta_1), t(beta_2)
sage: Q = C + plot(beta_1*M, width=3, color=image_color)
sage: Q = Q + plot(delta_1,width=1.4,color='blue')
sage: Q = Q + plot(beta_2*M,width=3,color=image_color)
sage: Q = Q + plot(delta_2,width=1.4,color='blue')
sage: Q.save("sageoutput/svd02b.png", figsize=4.467, axes_pad=0.05)
```

In the picture below, the domain's blue $\vec{\beta}$'s on the left map to the light red vectors on the right. Also on the right, in blue, are the $\vec{\delta}$'s. The red $t(\vec{\beta}_1)$ does look to be about 5.5 times $\vec{\delta}_1$, and $t(\vec{\beta}_2)$ does look something like 0.4 times $\vec{\delta}_2$.



(∗∗)

Note also that the two bases are *orthonormal*—the unit circles help us see that the bases are comprised of unit vectors, and further, the two members of each basis are orthogonal.

Compare this diagram to the one before it labelled (∗), which shows the effect of the matrix on the unit circle. We used the whole circle in (∗) to spotlight the ellipse, to make it easier to see that in (∗∗) the longest red vector is a semi-major axis of that ellipse and the shortest red vector is a semi-minor axis.

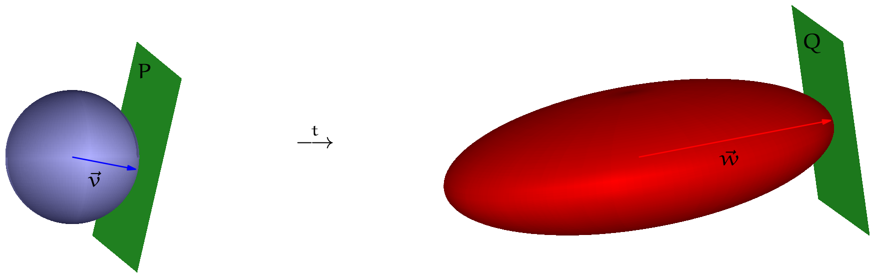
Proof sketch

This argument, adapted from Blank et al. [1989], is a sketch because it uses results that a typical reader has only seen in a less general version and because it relies on material from the book that is in an optional section. In addition, we'll consider only the case of a nonsingular matrix and map; it shows the main idea, which is the point of a sketch.

Consider an $n\times n$ matrix T that is nonsingular, and the nonsingular transformation $t\colon \mathbb{R}^n \to \mathbb{R}^n$ that is represented by T with respect to the standard bases.

Recall the Extreme Value Theorem from Calculus I: for a continuous function f, if $D \subset \mathbb{R}$ is a closed and bounded set then the image $f(D)$ is also a closed and bounded set (see Wikipedia [2012a]). A generalization of that result gives that because the unit sphere in $\mathbb{R}^n$ is closed and bounded then its image under t is closed and bounded. The image is in fact an ellipsoid, although we won't prove it, so we will call it that.

Because the ellipsoid is closed and bounded it has a point furthest from the origin. Let $\vec{w}$ be a vector extending from the origin to that furthest point. Let $\vec{v}$ be the member of the unit sphere that maps to $\vec{w}$. Let P be the plane that touches the sphere only at the endpoint of $\vec{v}$. Let Q be the image of P under t. Since t is one-to-one, Q touches the ellipsoid only at $\vec{w}$.



The tangent plane P has the property that the set of vectors whose bodies lie in P are the set of vectors perpendicular to $\vec{v}$. That is, if in the picture above we slide P along the vector $\vec{v}$ to the origin then we have the subspace of $\mathbb{R}^n$ of vectors perpendicular to $\vec{v}$. This subspace has dimension $n-1$. We will argue below that if we similarly slide Q to the origin then we have the set of vectors perpendicular to $\vec{w}$. With that we will have an argument by induction: we start constructing the bases B and D by taking $\vec{\beta}_1$ to be $\vec{v}$, taking σ_1 to be the length $\|\vec{w}\|$, and taking $\vec{\delta}_1$ to be $\vec{w}/\|\vec{w}\|$.

So consider Q. First observe that there is a unique plane that touches the ellipsoid only at $\vec{w}$ since if there were another $\hat{Q}$ then $t^{-1}(\hat{Q})$ would be a second plane that only touches the sphere at $\vec{v}$, which is impossible. To see that Q is perpendicular to $\vec{w}$ consider a sphere centered at the origin whose radius is $\|\vec{w}\|$. This sphere has a plane tangent at the endpoint of $\vec{w}$, which is perpendicular to $\vec{w}$. Because $\vec{w}$ ends at a point on the ellipsoid furthest from the origin, the ellipsoid is entirely contained in the sphere, so this plane touches the ellipsoid only at $\vec{w}$. Therefore, from the second sentence of this paragraph, this plane is Q. That ends the proof.

Matrix factorization

We can express those geometric ideas in an algebraic form (for a proof see Trefethen and Bau [1997]).

The *singular value decomposition* of an $m \times n$ matrix A is a factorization $A = U\Sigma V^T$. The $m \times n$ matrix Σ is is all zeroes except for diagonal entries, the singular values, $\sigma_1 \geqslant \sigma_2 \geqslant \cdots \geqslant \sigma_r > 0$ where r is the rank of A. The $m \times m$ matrix U and the $n \times n$ matrix V are unitary, meaning that their columns form an orthogonal basis of unit vectors, the left and right *singular vectors* for A, respectively.

```
sage: M = matrix(RDF, [[0, 1, 2], [3, 4, 5]])
sage: U,Sigma,V = M.SVD()
sage: U
```

```
[-0.274721127897  -0.961523947641]
[-0.961523947641   0.274721127897]
sage: Sigma
[7.34846922835              0.0            0.0]
[            0.0            1.0            0.0]
sage: V
[-0.392540507864   0.824163383692   0.408248290464]
[-0.560772154092   0.137360563949  -0.816496580928]
[  -0.72900380032  -0.549442255795   0.408248290464]
```

The product $U\Sigma V^T$ simplifies. To see how, consider the case where all three matrices are $2{\times}2$. Write $\vec{u}_1$, $\vec{u}_2$ for the columns of U and $\vec{v}_1$, $\vec{v}_2$ for the columns of V, so that the rows of V^T are $\vec{v}_1^T$ and $\vec{v}_2^T$.

$$\begin{aligned}
U\Sigma V^T &= \begin{pmatrix}\vec{u}_1 & \vec{u}_2\end{pmatrix}\begin{pmatrix}\sigma_1 & 0\\ 0 & \sigma_2\end{pmatrix}\begin{pmatrix}\vec{v}_1^T\\ \vec{v}_2^T\end{pmatrix}\\
&= \begin{pmatrix}\vec{u}_1 & \vec{u}_2\end{pmatrix}\begin{pmatrix}\sigma_1 & 0\\ 0 & 0\end{pmatrix}\begin{pmatrix}\vec{v}_1^T\\ \vec{v}_2^T\end{pmatrix} + \begin{pmatrix}\vec{u}_1 & \vec{u}_2\end{pmatrix}\begin{pmatrix}0 & 0\\ 0 & \sigma_2\end{pmatrix}\begin{pmatrix}\vec{v}_1^T\\ \vec{v}_2^T\end{pmatrix}\\
&= \sigma_1\cdot\begin{pmatrix}\vec{u}_1 & \vec{u}_2\end{pmatrix}\begin{pmatrix}1 & 0\\ 0 & 0\end{pmatrix}\begin{pmatrix}\vec{v}_1^T\\ \vec{v}_2^T\end{pmatrix} + \sigma_2\cdot\begin{pmatrix}\vec{u}_1 & \vec{u}_2\end{pmatrix}\begin{pmatrix}0 & 0\\ 0 & 1\end{pmatrix}\begin{pmatrix}\vec{v}_1^T\\ \vec{v}_2^T\end{pmatrix} \qquad (***)
\end{aligned}$$

In the first term right multiplication by the $1,1$ unit matrix picks out the first column of U and left multiplication by the $1,1$ unit matrix picks out first row of V so those are the only parts that remain after the product. In short, we get this.

$$\begin{pmatrix}u_{1,1} & u_{1,2}\\ u_{2,1} & u_{2,2}\end{pmatrix}\begin{pmatrix}1 & 0\\ 0 & 0\end{pmatrix}\begin{pmatrix}v_{1,1} & v_{2,1}\\ v_{1,2} & v_{2,2}\end{pmatrix} = \begin{pmatrix}u_{1,1}v_{1,1} & u_{1,1}v_{2,1}\\ u_{2,1}v_{1,1} & u_{2,1}v_{2,1}\end{pmatrix} = \vec{u}_1\vec{v}_1^T$$

Thus, equation $(***)$ simplifies to $U\Sigma V^T = \sigma_1\cdot\vec{u}_1\vec{v}_1^T + \sigma_2\cdot\vec{u}_2\vec{v}_2^T$. Cases other than $2{\times}2$ simplify in the same way.

Application: data compression

We now can write any matrix as a sum $M = \sigma_1\cdot\vec{u}_1\vec{v}_1^T + \sigma_2\cdot\vec{u}_2\vec{v}_2^T + \cdots$ where the vectors have unit size and the σ_i's decrease in size.

Suppose that the matrix is $n{\times}n$. If we need to store it or transmit it then we need to work with n^2 real numbers. For instance, if $n = 500$ then we have $50^2 = 250\,000$ reals. If we instead express the matrix as a sum then each term requires n real numbers for $\vec{u}_i$, another n reals for $\vec{v}_i$, and one more real for σ_i. So saving all the terms would require $500 * (2 * 500 + 1) = 500\,500$ reals, which is not a savings. But, if we store only some of the terms, say the first 50 of them then it would be a savings: $50\cdot(2n+1) = 50\,050$, which is about 20% of the 500^2 size of the full matrix. Thus, if we have data as a matrix then we could hope to compress it with that formula by dropping terms with small σ's. The question is whether we lose too much of the the information by only retaining the information associated with 10% of the singular values.

We will illustrate with image compression. Meet Lenna. This top third of a pinup is a standard test image Wikipedia [2012b].

We will use the Python Image Library for help in reading the figure, getting the color value at each pixel, etc. The code is in a function that is listed at the end of this chapter, `img_squeeze`. It takes three arguments, the names of the input and output functions, and a real number between 0 and 1 which determines the percentage of the singular values to include in the sum before the cutoff.

This function takes the input data into a format where each pixel is a triple (red, green, blue) of integers that range from 0 to 255. It uses those numbers to build three Python arrays `rd`, `gr`, and `bl`, which then initialize three Sage matrices `RD`, `GR`, and `BL`.

The next step finds the Singular Value Decomposition of those three. Just out of curiosity, we have a peek at the eight largest singular values in the red matrix, the singular value where we make the cutoff, and the eight smallest.

```
sigma_RD 0 = 93233.7882139
sigma_RD 1 = 11193.1186308
sigma_RD 2 = 8660.12549856
sigma_RD 3 = 7054.38662024
sigma_RD 4 = 5916.89456484
sigma_RD 5 = 5742.14618029
sigma_RD 6 = 4125.16604075
sigma_RD 7 = 3879.15303685
  :
sigma_RD 51 = 606.389279688
  :
sigma_RD 504 = 0.547288023466
sigma_RD 505 = 0.470266870056
sigma_RD 506 = 0.1988515109
sigma_RD 507 = 1.02472345283e-11
sigma_RD 508 = 1.02472345283e-11
sigma_RD 509 = 1.02472345283e-11
sigma_RD 510 = 1.02472345283e-11
sigma_RD 511 = 2.32533986491e-12
```

The large singular values are much larger than the small singular values, even setting aside the ones at the end that are zero except for numerical issues.

Finally, for each matrix we compute the sum $\sigma_1 \cdot \vec{u}_1\vec{v}_1^{\mathsf{T}} + \sigma_2 \cdot \vec{u}_2\vec{v}_2^{\mathsf{T}} + \cdots$ up through some cutoff index.[1]

We want to see how badly the image degrades for various cutoffs. The tradeoff is that each term costs about $2 \cdot 512$ real numbers to store or transmit and we want to substantially beat the original image's $512 \cdot 512$. Thus we want to give the function a parameter of a good bit less than 50%.

The code below sets it to 10%. This image is 512×512 so that will sum the terms associated with the first 51 singular values.[2]

```
sage: load "img_squeeze.sage"
sage: img_squeeze("Lenna.png", "Lena_squeezed.png", 0.10)
```

Below is the squeezed image, with the contributions of 10% of the singular values. The image quality is less than the original. The colors are not as good, the edges are not sharp, and there are a few artifacts, including a horizontal line across the top of Lenna's forehead and another halfway up her hat. But certainly the image is entirely recognizable. Again, the advantage is that this image takes only 20% of the storage or transmission requirements.

Experimenting with the percentage value shows that setting the parameter to 20% is enough to make the output image hard to tell from the original.

[1]The code asks for the transpose of the `U_RD.column(i)`, etc., which may seem to be a mistake. Remember that Sage favors rows for vectors, so this is how we make the $\vec{u}$ vector a column, while $\vec{v}$ is already a row. Because of this operation on the vector, the first time you run this code in a Sage session you may get a depreciation warning. [2]The function takes a very long time to run, in part because the code is intended to be simple rather than fast. Consequently, in the function source there are some status lines that help convince a person executing the code that it is still working; here that busy-work output is omitted.

Source of plot_action.sage

```
# plot_action.sage
# Show the action of a 2x2 matrix on the top half of a unit circle

DOT_SIZE = .02
```

```
CIRCLE_THICKNESS = 2
def color_circle_list(a, b, c, d, colors, full_circle = False):
    """Return list of graph instances for the action of a 2x2 matrix on
    half of the unit circle.  That circle is broken into chunks each
    colored a different color.
      a, b, c, d  reals  entries of the matrix
      colors  list of rgb tuples; len of this list is how many chunks
      full_circle=False  Show a full circle instead of a half circle.
    """
    r = []
    if full_circle:
        p = 2*pi
    else:
        p = pi
    t = var('t')
    n = len(colors)
    for i in range(n):
      color = colors[i]
        x(t) = a*cos(t)+b*sin(t)
        y(t) = c*cos(t)+d*sin(t)
        g = parametric_plot((x(t), y(t)),
                            (t, p*i/n, p*(i+1)/n),
                            color = color, thickness=CIRCLE_THICKNESS)
      r.append(g)
        r.append(circle((x(p*i/n), y(p*i/n)), DOT_SIZE, color=color))
    if not(full_circle):    # show (x,y)=(-1,0) is omitted
        r.append(circle((x(pi), y(pi)), 2*DOT_SIZE, color='black',
                        fill = 'true'))
        r.append(circle((x(pi), y(pi)), DOT_SIZE, color='white',
                        fill = 'true'))
    return r

def plot_circle_action(a, b, c, d, n = 12, full_circle = False, show_unit_circle =
    """Show the action of the matrix with entries a, b, c, d on half
    of the unit circle, as the circle and the output curve, broken into
    a number of colors.
     a, b, c, d  reals  Entries are upper left, ur, ll, lr.
     n = 12  positive integer  Number of colors.
    """
    colors = rainbow(n)
    G = Graphics()  # holds graph parts until they are to be shown
    if show_unit_circle:
        for f_part in color_circle_list(1,0,0,1,colors,full_circle):
            G += f_part
    for g_part in color_circle_list(a,b,c,d,colors,full_circle):
        G += g_part
    return plot(G)

SQUARE_THICKNESS = 1.75  # How thick to draw the curves
ZORDER = 5    # Draw the graph over the axes
def color_square_list(a, b, c, d, colors):
    """Return list of graph instances for the action of a 2x2 matrix
    on a unit square.  That square is broken into sides, each colored a
    different color.
      a, b, c, d  reals  entries of the matrix
      colors  list of rgb tuples; len of this list is at least four
    """
    r = []
    t = var('t')
    # Four sides, ccw around square from origin
    r.append(parametric_plot((a*t, b*t), (t, 0, 1),
                             color = colors[0], zorder=ZORDER,
                             thickness = SQUARE_THICKNESS))
    r.append(parametric_plot((a+c*t, b+d*t), (t, 0, 1),
                             color = colors[1], zorder=ZORDER,
                             thickness = SQUARE_THICKNESS))
    r.append(parametric_plot((a*(1-t)+c, b*(1-t)+d), (t, 0, 1),
                              color = colors[2], zorder=ZORDER,
                              thickness = SQUARE_THICKNESS))
    r.append(parametric_plot((c*(1-t), d*(1-t)), (t, 0, 1),
                             color = colors[3], zorder=ZORDER,
                             thickness = SQUARE_THICKNESS))
    # Dots make a cleaner join between edges
    r.append(circle((a, b), DOT_SIZE,
                    color = colors[0], zorder = 2*ZORDER,
                    thickness = SQUARE_THICKNESS*1.25, fill =  True))
    r.append(circle((a+c, b+d), DOT_SIZE,
                    color = colors[1], zorder = 2*ZORDER+1,
                    thickness = SQUARE_THICKNESS*1.25, fill =  True))
    r.append(circle((c, d), DOT_SIZE,
                    color = colors[2], zorder = ZORDER+1,
                    thickness = SQUARE_THICKNESS*1.25, fill =  True))
    r.append(circle((0, 0), DOT_SIZE,
                    color = colors[3], zorder = ZORDER+1,
                    thickness = SQUARE_THICKNESS*1.25, fill =  True))
    return r

def plot_square_action(a, b, c, d, show_unit_square = False):
    """Show the action of the matrix with entries a, b, c, d on half
    of the unit circle, as the circle and the output curve, broken into
    colors.
     a, b, c, d  reals  Entries are upper left, ur, ll, lr.
    """
    colors = ['red', 'orange', 'green', 'blue']
    G = Graphics()        # hold graph parts until they are to be shown
    if show_unit_square:
        for f_part in color_square_list(1,0,0,1,colors):
            G += f_part
    for g_part in color_square_list(a,b,c,d,colors):
        G += g_part
    p = plot(G)
    return p
```

```

plot.options['figsize'] = 2.5
plot.options['axes_pad'] = 0.05
plot.options['fontsize'] = 7
plot.options['dpi'] = 500
plot.options['aspect_ratio'] = 1
```

Source of img_squeeze.sage

```
# Compress the image
import Image

def img_squeeze(fn_in, fn_out, percent):
    """Squeeze an image using Singular Value Decomposition.
        fn_in, fn_out  string  name of file
        percent  real in 0..1  Percentage of total singular values to use
    """
    img = Image.open(fn_in)
    img = img.convert("RGB")
    rows, cols = img.size
    cutoff = int(round(percent*min(rows,cols),0))
    # Gather data into three arrays, then give to Sage's matrix()
    rd, gr, bl = [], [], []
    for row in range(rows):
        for a in [rd, gr, bl]:
           a.append([])
        for col in range(cols):
            r, g, b = img.getpixel((int(row), int(col)))
            rd[row].append(r)
            gr[row].append(g)
            bl[row].append(b)
    RD, GR, BL = matrix(RDF, rd), matrix(RDF, gr), matrix(RDF, bl)
    # Get the SVDs
    print "about to get the svd"
    U_RD,Sigma_RD,V_RD = RD.SVD()
    U_GR,Sigma_GR,V_GR = GR.SVD()
    U_BL,Sigma_BL,V_BL = BL.SVD()
    # Have a look
    for i in range(8):
        print "sigma_RD",i,"=",Sigma_RD[i][i]
    print "  :"
    print "sigma_RD",cutoff,"=",Sigma_RD[cutoff][cutoff]
    print "  :"
    for i in range(Sigma_RD.nrows()-8,Sigma_RD.nrows()):
        print "sigma_RD",i,"=",Sigma_RD[i][i]
    # Compute sigma_1 u_1 v_1^trans+ ..
    a=[]
    for i in range(rows):
        a.append([])
        for j in range(cols):
            a[i].append(0)
    A_RD, A_GR, A_BL = matrix(RDF, a), matrix(RDF, a), matrix(RDF, a)
    for i in range(cutoff):
        sigma_i = Sigma_RD[i][i]
        u_i = matrix(RDF, U_RD.column(i).transpose())
        v_i = matrix(RDF, V_RD.column(i))
        A_RD = copy(A_RD)+sigma_i*u_i*v_i
        sigma_i = Sigma_GR[i][i]
        u_i = matrix(RDF, U_GR.column(i).transpose())
        v_i = matrix(RDF, V_GR.column(i))
        A_GR = copy(A_GR)+sigma_i*u_i*v_i
        sigma_i = Sigma_BL[i][i]
        u_i = matrix(RDF, U_BL.column(i).transpose())
        v_i = matrix(RDF, V_BL.column(i))
        A_BL = copy(A_BL)+sigma_i*u_i*v_i
    # Make a new image
    img_squoze = Image.new("RGB", img.size)
    print "fn_out is ",fn_out
    for row in range(rows):
        print "transferring over row=", row
        for col in range(cols):
            p = (int(A_RD[row][col]), int(A_GR[row][col]), int(A_BL[row][col]))
            img_squoze.putpixel((int(row), int(col)), p)
    print "about to save: ",fn_out
    img_squoze.save(fn_out)
```

Bibliography

Robert A. Beezer. Sage for Linear Algebra. http://linear.ups.edu/download/fcla-2.22-sage-4.7.1-preview.pdf, 2011.

S. J. Blank, Nishan Krikorian, and David Spring. A geometrically inspired proof of the singular value decomposition. *The American Mathematics Monthly*, pages 238–239, March 1989.

Jim Hefferon. Linear Algebra. http://joshua.smcvt.edu/linearalgebra, 2012.

David Joyner and William Stein. Open source mathematical software. *Notices of the AMS*, page 1279, November 2007.

Ron Brandt. *Powerful Learning*. Association for Supervision and Curriculum Development, 1998.

Sage Development Team. Sage tutorial 5.3. http://www.sagemath.org/pdf/SageTutorial.pdf, 2012a.

Sage Development Team. Sage reference manual 5.3. http://www.sagemath.org/pdf/reference.pdf, 2012b.

Shunryu Suzuki. *Zen Mind, Beginners Mind*. Shambhala, 2006.

Lloyd N. Trefethen and David Bau, III. *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, 1997.

Wikipedia. Extreme value theorem, 2012a. URL http://en.wikipedia.org/wiki/Extreme_value_theorem. [Online; accessed 28-Nov-2012].

Wikipedia. Lenna, 2012b. URL http://en.wikipedia.org/wiki/Lenna. [Online; accessed 29-Nov-2012].